CrossGuardian: A Security Domain Isolation Model for Cross-Chain Gateway

Haosu Cheng, Guanquan Shi, and Kangkang Zhang

***Abstract*—Blockchain, as a distributed ledger technology, embodies decentralization, security transparency, and traceability. Interoperability has been a persistent challenge in blockchain technology research. The inherent characteristics of blockchain, including numerous terminal nodes, diverse network architectures, cross-chain application services, and intricate network environments, present significant privacy and data security concerns for cross-chain gateways during data collection and transmission. To address these security challenges, this paper proposes a system architecture and security domain isolation model for cross-chain security gateways leveraging system resource virtualization. This architecture aims to address security issues encountered in the process of collecting, transmitting, and exchanging data elements within the cross-chain gateway. Specifically, the proposed architecture isolates the different layers in the cross-chain gateway by business domains, and ensures mutual security isolation between distinct business domains and network domains during cross-chain activities. Moreover, the paper implements a secure gateway operational environment utilizing x86 hardware platforms and virtualization technology, followed by simulation experiment and performance testing to validate the architectural feasibility. Formal proof is provided to establish the security of the proposed domain model.**

***Index Terms*—Blockchain cross-chain, cross-chain gateway, cross-domain services, security domain isolation.**

I. Introduction

As blockchain technology advances, its integration spans across various industries such as finance, IoT, logistics, healthcare, insurance, public administration, among others [1]. This trend has led to the emergence of diverse types of blockchains, categorized based on authorization or consensus algorithms [2]. Each blockchain network typically operates independently, lacking direct interaction with the external world and the ability to transfer assets or data with each other. Oracle technology [3] facilitates off chain data connectivity for smart contracts, ensuring blockchain to communicate with real-world data. Meanwhile, cross-chain technology enables interoperability among different blockchain networks, ensuring the sharing of homogeneous or heterogeneous blockchain data [4].

As shown in the Fig. 1, the cross-chain gateway is a technology or protocol of the cross-chain scheme Sidechains/Relays, effectively promoting the exchange of data, assets or information between different blockchains. When handling cross-chain transactions, gateways aggregate transactions from various application chains and the relay chain, characterized by different types and security levels. This leads to the complexity of resource deployment in cross-chain gateways. If the deployment is based on a low-security blockchain application, the high-security application will not meet its security requirements. If the deployment is based on high security, the resources will be wasted and additional resource overhead will be generated. Moreover, existing cross-chain gateways in upholding user privacy and security [5] suffer from various security threats, primarily due to three critical challenges:

1. Security Downgrade: Shared resources among heterogeneous modules (e.g., consensus engines, smart contracts) allow vulnerabilities in one component to compromise the entire system. For instance, a flawed signature verification in an Ethereum plugin could enable attackers to forge cross-chain transactions.

2. Data Leakage: Sensitive information (e.g., private keys, transaction metadata) is exposed during transmission or storage, as current architectures lack encryption and access control between modules. A notable example is the interception of unencrypted inter-blockchain transmission protocol (IBTP) packets in relay chains.

3. Inter-Chain Attack Propagation: Malicious actors exploit vulnerabilities in one blockchain (e.g., a consensus flaw) to propagate attacks to interconnected chains via the gateway. The 2021 Poly Network hack demonstrated how compromised smart contracts could drain assets across multiple chains.

The shared operational environment exacerbates these risks, as flaws in one module can compromise the integrity of the entire system, enabling cross-chain attacks or manipulation of transaction data. These challenges underscore the urgent need for a security model that enforces strict isolation while maintaining interoperability. Gateway endpoints must implement measures to segregate and fortify data with distinct security requisites within cross-chain scenarios.

However, existing gateway terminals within cross-chain networks exhibit deficiencies in capabilities such as transaction information isolation, cross-chain verification informa-

Manuscript received August 21, 2024; revised March 27, 2025; approved for publication by Wen, Wanli, Division 3 Editor, June 3, 2025.

This paper is supported by the National Key Research and Development Program of China (2023YFB2703903), the Shandong Innovation Capability Improvement Project for Technological SMEs (2022TSGC2044) and the Shandong Hi-Speed Group Technology Innovation Project (HSB2021-15).

H. Cheng is with Shandong Key Laboratory of Blockchain Finance, Shandong University of Finance and Economics, Jinan, China; School of Software, Shandong University, Jinan, China; Shandong CVIC Software Engineering Co., Ltd, Jinan, China, email:chenghaosu@sdufe.edu.cn.

G. Shi and K. Zhang are with Shandong Key Laboratory of Blockchain Finance, Shandong University of Finance and Economics, Jinan, China; School of Computer Science and Technology, Shandong University of Finance and Economics, Jinan, China; Shandong Technology Innovation Center of Social Governance Intelligence, Shandong University of Finance and Economics, Jinan, China, email:{222115010, zhangkk}@mail.sdufe.edu.cn.

K. Zhang is the corresponding author.

Digital Object Identifier: 10.23919/JCN.2025.000037

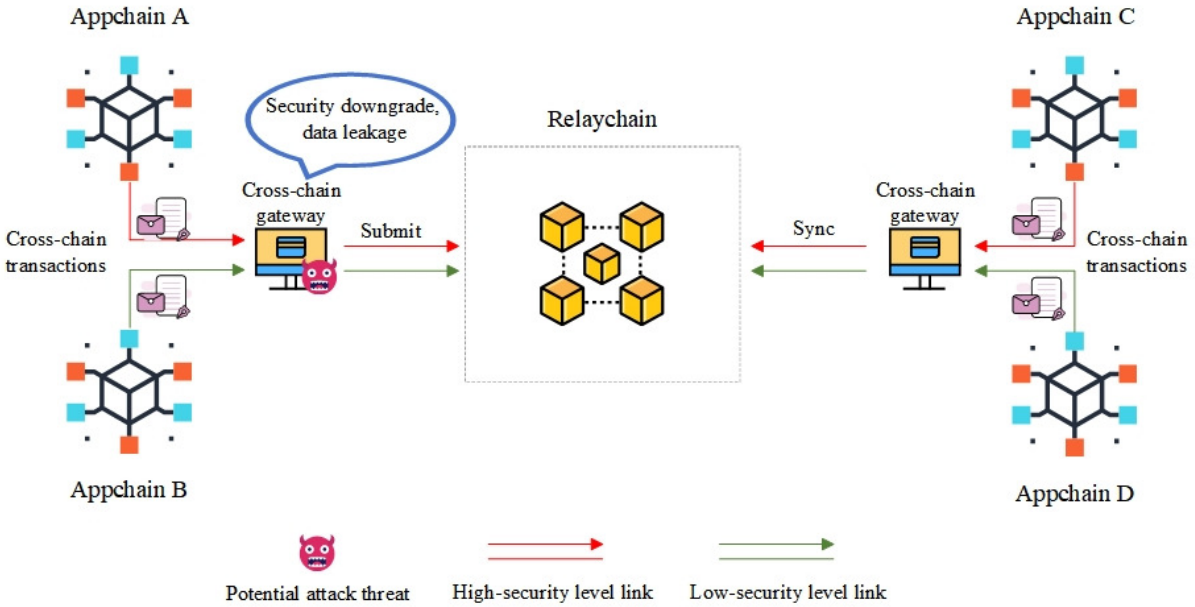

Fig. 1. Security risks that arise from cross-chain gateways during cross-chain transactions.

tion isolation, and cross-chain bridge information isolation. The frequent authentication switches upon entry into diverse blockchains may result in operational disruptions, failing to align with the flexible and dynamic access requisites across various application contexts. Moreover, inherent security risks associated with gateway terminal devices, coupled with behaviors such as cross-domain communication, pose threats of boundary breaches and information leakage [6]. Current the majority of research is centered around protocols and technical solutions to ensure the reliability of cross-chain transactions, with a lack of studies focusing on addressing these threats from the architectural level of the cross-chain gateway system.

The primary contributions of this paper can be summarized as follows:

1) We introduce an isolation model virtualized resource-based system domain isolation model (VRSDIM) grounded in hardware resource virtualization. The security assertions of this proposed domain isolation model are rigorously substantiated through formal verification processes.
2) The cross-chain gateway security isolation model (CrossGuardian) based on VRSDIM technology is designed for cross-chain gateway system architecture. The model realizes the security isolation between the application chain layer, the interaction layer and the relay layer in the cross-chain gateway, and provides a secure deployment environment conducive to data processing, cross-chain transaction transmission and various blockchain network interfaces.
3) We established the CrossGuardian on the x86 hardware platform, in which we conducted a simulation experiment of processing transactions in the cross-network gateway. In addition, through various benchmarks we evaluated the performance of virtual resource process scheduling, security services, and network functions in this model. These comprehensive evaluations validate the viability of the CrossGuardian architecture.

II. Related Work

Blockchain is a decentralized, immutable, pseudonymous, and transparently verifiable distributed ledger, organized on the chronological sequencing of data blocks. Leveraging peer-to-peer (P2P) networks, data encryption, timestamps, distributed consensus mechanisms, and incentivization protocols, blockchain nodes within the network can participate in peer-to-peer transactions. Due to diverse industry-specific requirements, different and decentralized application chains has been produces, resulting in the emergence of numerous ”data islands” [7]. Furthermore, owing to the different consensus algorithms and data formats of blockchains, direct access to data across different chains is often unattainable. Crosschain technology serves as a pivotal facilitator, effectively activating and interlinking established blockchains, particularly those with entrenched and immutable characteristics. This enables secure implementation of cross-system and cross-domain business processes [8].

Notary Schemes (External Certification). These schemes rely on external validators for verification, and can be categorized into two types: single-point and multi-point validation. Single-point validation [9] involves only one external validator. In contrast, multi-point validation [10] utilizes multiple external validators, mitigating the risk of single-point attacks and promoting decentralization compared to single-point validation.

Hash-locking (Local Certification). The hashed timeLock contract (HTLC) [11] is a protocol implemented at the smart contract layer, commonly referred to as a hash time lock contract. The core components of HTLC are the hash lock and time lock. The time lock ensures that a transaction is

valid only if submitted within an agreed timeframe; if the time expires, the commitment scheme becomes invalid (regardless of whether it was initiated by the proposer or the recipient). The hash lock requires that the original image R corresponding to the hash value $Hash(R) = H$ is provided to execute the transaction and fulfill the commitment.

Sidechains/Relays (Native Certification). Inter-chain interactions require a smart contract capable of accessing the main chain's network data. This cross-chain protocol includes methods for data switching mechanisms between different chains, enabling interactions between the main chain and various sidechains through smart contracts [12]. Different chains perform native cross-chain validation via light clients, supporting two types of cross-chain methods: homogeneous cross-chain and heterogeneous cross-chain.

Distributed Private Key Control. Distributed private key control is a method based on cryptographic techniques such as multiparty computation and threshold key technology. It achieves secure control of assets within a blockchain system by splitting the private key and storing it across multiple independent nodes. This approach separates the usage rights from the ownership of digital assets, enabling the secure transfer of asset control from centralized systems to decentralized networks. Additionally, distributed private key control allows on-chain assets to be mapped onto cross-chain systems, facilitating asset circulation and value transfer between different blockchains [13].

Layer-Zero [14] leverages Oracle and Relayer as third-party notaries for transmitting cross-chain data to LayerZero Endpoint smart contracts, which subsequently interact with contracts across diverse chains. In contrast, Rainbow Bridge utilizes Relayer for cross-chain data transmission and conducts native cross-chain verification via peer-to-peer light clients. Prominent cross-chain projects like Polkadot and BitXHub employ relay chain technology to achieve data and business interconnection and interoperability. Polkadot [15] employs the ICMP protocol for cross-chain communication. When Chain A initiates a cross-chain transaction, collectors aggregate the transaction and submit it to Chain A's validators. Upon validation, the transaction is propagated to the relay chain, which confirms blocks and routes cross-chain transactions, ultimately leading to Chain B receiving and executing the cross-chain transaction block. BitXHub [16] encompasses three roles: application chain, relay chain, and cross-chain gateway. The relay chain and multiple application chains form a consortium, employing the IBTP universal cross-chain protocol for routing and verifying cross-chain transactions.

Zhang *et al.* [17] have proposed a cross-chain system model comprising users, application chains, and cross-chain networks. Through the cross-chain network, users can dispatch transaction requests to the designated addresses of respective application chains. Gateways embedded within the cross chain network are responsible for managing the transmission of transaction requests, monitoring transaction events initiated by application chains, and upon receiving such events, placing transactions into a pending transaction pool. However, in the management of disparate security levels and varied types of business and data, cross-chain gateways are susceptible to data attacks and information leakage risks [18], including Double Spending Attacks [19], Collusion Attacks [20], Single Point Failures [21], and Private Key Attacks [19]. Failure to isolate security between chains may result in the entire cross-chain network being compromised if one chain falls victim to an attack. Typically, security is ensured through three main avenues: appropriate isolation, security event detection, and validation of cross chain transaction correctness [22]. Chains are advised to uphold their autonomy and process cross-chain transactions through third-party nodes or independent modules equipped with secure isolation. This approach guarantees that issues arising from cross-chain transactions do not affect the processing of on-chain transactions. With third-party nodes or independent modules equipped with the capability to detect and respond to security events, the architectural isolation of the system is further reinforced, akin to the functionality of a firewall within the cross-chain protocol or system.

Common threat models for blockchain cross-chain gateways encompass:

1. Untrusted Firmware Upgrade [23]: Attackers may exploit vulnerabilities or insecure communication channels during the firmware upgrade process to inject malicious code or tamper with legitimate firmware on a cross-chain gateway. This can lead to remote control of the cross-chain gateway, data theft, or denial-of-service attacks, thereby compromising the security of the system.

2. Inter-Chain Attack Propagation [24]: Attackers may exploit vulnerabilities or attack vectors on one blockchain to attempt to compromise the security and stability of another blockchain or the cross-chain gateway. This can lead to the spread of attacks across multiple chains, increasing the overall risk to the interconnected blockchain network.

3. Unauthorized Access [25]: Attackers may gain unauthorized access to sensitive data, tamper with transaction records, execute malicious smart contracts, or perform other unauthorized actions. This can lead to significant security breaches, compromising the integrity and confidentiality of the blockchain system.

4. Data Leakage [8]: Attackers may obtain or expose sensitive data within a cross-chain gateway through malicious cyberattacks, security vulnerabilities, or insecure data transmission and storage methods. This can lead to significant security risks, including unauthorized access to confidential information and potential misuse of the compromised data.

Security isolation [26] is a method employed to partition distinct components of a computer system or network, either physically or logically, to mitigate the propagation of malicious activities or errors within the system. Techniques encompassed within security isolation include virtualization, containerization, network segmentation, access control mechanisms, and enforcement of security policies, among others. These measures aim to confine the access boundaries of sensitive data or system functionalities, thereby mitigating the likelihood of system compromise or disruption and enhancing overall security and reliability. Within Table 1, we enumerate five security isolation methodologies and provide a condensed overview of their defense against various threat models.

Within Table 1, we enumerate five security isolation

TABLE I
CROSS-CHAIN SECURITY ISOLATION TECHNOLOGY COMPARISON.

Virtualization technology	Non-trusted firmware upgrade	Inter-chain attack propagation	Unauthorized access	Data leakage
Decentralized-KPD [27]	-	✓	✓	-
FlexOS [28]	-	-	✓	✓
Donky [29]	✓	✓	✓	✓
Hardware-Based Domain Virtualization [30]	✓	✓	✓	✓
Our work	✓	✓	✓	✓

methodologies and provide a condensed overview of their defense against various threat models.

Decentralized-KPD [27] leverages hardware virtualization and address remapping technologies to implement distinct security mechanisms across multiple protection domains. Its security assumptions primarily address attacks targeting vulnerabilities in security mechanism code within functional protection domains, while overlooking denial-of-service attacks and physical layer assaults. As a result, while this solution effectively combats the propagation of inter-chain attacks and unauthorized access, it lacks robust defenses against unauthorized firmware upgrades and data leakage. FlexOS [28] enables users to mix and match isolation primitives, enabling the creation of customized application versions tailored to specific security requirements, effectively mitigating unauthorized access and data leakage. However, the porting process entails not only internal kernel library porting but also external user-space library porting. While this is a one-time operation with relatively minimal costs, it may lead to limited or excessive data sharing, diminishing flexibility and potentially compromising defense against unauthorized firmware upgrades and inter-chain attack proliferation. Donky [29] relies on a small memory protection key hardware extension to facilitate the DonkyLib software framework, achieving secure and efficient inter-process isolation. Leveraging hardware domain virtualization technology [30], two architectural schemes—hardware-based MPK virtualization and hardware based domain virtualization—are proposed to partition different execution environments into separate virtualization domains via hardware-level isolation. While both isolation technologies effectively mitigate various threat models, they are developed at the hardware level. Compared to virtualization-based security domain isolation models, these hardware-level solutions incur higher costs and offer less flexibility. Virtualization-based security domain isolation models provide a system resource empowerment platform at the operating system level. By utilizing a hypervisor to partition physical servers into multiple isolated virtual environments, each virtual environment operates on a virtual server and inherits all functions of the physical server [31]. This distributed architecture ensures isolated control over information exchange between different business domains within the device system, safeguarding against any failure or security vulnerability in one business domain affecting other partitions. Virtualization-based security domain isolation models feature straightforward migration and robust flexibility, while also accommodating multiple chip architectures.

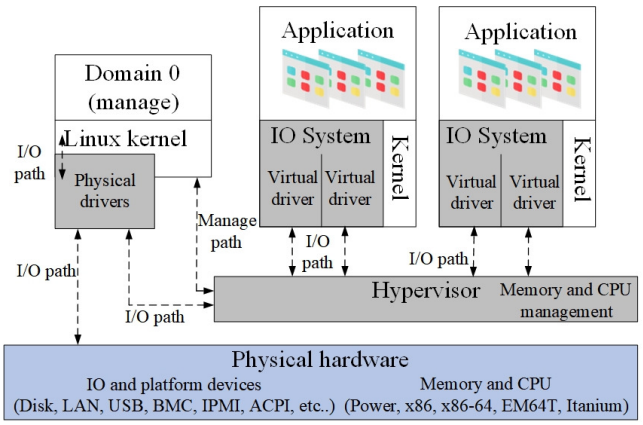

Fig. 2. The architecture of the VRSDIM.

In summary, current resource virtualization technologies primarily cater to hardware resource mapping, isolation, allocation, and management within cloud environments. However, they lack tailored considerations and designs for future emerging applications such as blockchain terminal environments, data element collection, and cross-chain transactions. Particularly, there is a deficiency in addressing system resource configuration management and security isolation from the perspective of intersecting network domains and business domains. Consequently, existing blockchain gateway systems fall short in meeting the functional requirements for isolating blockchain data between distinct business domains and network domains.

III. SECURITY DOMAIN ISOLATION MODEL BASED ON VIRTUAL RESOURCES

In this chapter, our focus lies on the exposition of a security domain isolation model rooted in virtualization principles, augmented by the integration of formal verification techniques. Section A describes the architecture of the VRSDIM, Section B performs a formal analysis on the computational overhead of the architecture, Section C elucidates essential definitions pertinent to the model, while Section D employs a formal proof framework [32] to substantiate the security claims inherent to the model.

A. The Architecture of VRSDIM

As illustrated in Fig. 2, in the VRSDIM architecture, the VMM (hypervisor) sits between the operating system and the hardware, providing virtualization management and allocation of hardware resources while ensuring the secure isolation of virtual machine instances (i.e., between virtual domains). The VRSDIM model adopts a hybrid virtualization model, which assists the hypervisor by setting a privileged management domain (Domain 0) to control other user domains (Domain U) and provide necessary virtualization resources.

Each domain is equipped with an abstraction layer that includes APIs for managing and interfacing with virtual hardware. Domain 0 (Dom0) contains native device drivers

that have direct access to physical hardware and support user-mode management tools, enabling unified management of the virtualized environment. VRSDIM also introduces a split device driver model. This model establishes front-end devices in each user domain (DomU) and back-end devices in the privileged domain (Dom0). The operating systems in user domains interact with front-end devices as if they were ordinary devices, sending requests through the front-end devices. These requests, along with the identity information of the user domains, are transmitted to the back-end devices in the privileged domain via I/O request descriptors (I/O descriptor rings) and device channels. This architecture separates the handling of control information from data transfer.

The privileged domain hosting the back-end devices is also referred to as the isolation device domain (IDD). All access to physical hardware is initiated by the back-end devices in the privileged domain through native device drivers. The front-end devices are designed to be highly simplified, responsible solely for forwarding data. As they are not actual device drivers, they do not need to perform request scheduling. Conversely, the back-end devices operating in the IDD leverage existing Linux device drivers to handle hardware access, requiring only the addition of an I/O request bridging function—capable of dispatching and returning tasks efficiently.

B. The analysis of computational overhead

In a virtualized environment, the computational overhead associated with hardware resource access primarily stems from several key factors, including the cost of context switching and privileged operations, performance bottlenecks in data transmission, and additional overhead introduced by memory management mechanisms.

1. I/O Virtualization (Network/Disk)

Based on the front-end drive separation model, the single I/O operation delay:

$$T_{I/O} = 2 \cdot T_{trap} + \frac{n}{B_{grant}} + T_{event}.$$

T_{trap}: The overhead of context switching into/returning to the hypervisor is O(1) in complexity, with the actual value determined by register save/restore operations (approximately 200–500 cycles) and the penalty induced by cache/TLB invalidation;

B_{grant}: Shared memory bandwidth (typical 10–50 GB/s);

T_{event}: Event channel atomic operation time (usually $<$ 100 ns), complexity O(1);

n: The amount of data transferred (bytes), the leading item is O(n).

2. Memory Virtualization (Page Table Management)

In paravirtualization (PV) mode, the virtual machine directly submits page table update requests. The cost of performing m operations:

$$T_{mmu} = m \cdot (T_{hypercall} + T_{validate}).$$

$T_{hypercall}$: Single Hypercall time (approximately 200–500 ns);

$T_{validate}$: Hypervisor Time to verify the validity of a page entry (O(1), approximately 50 ns).

3. Memory Overcommit

Let the overcommit ratio be defined as the follow.

$$\alpha = \frac{Total\ Allocated}{Physical\ Memory\ Capacity}.$$

When $\alpha > 1$ indicates memory overcommitment, the overhead of swapping out m pages is modeled as:

$$T_{swap} = m \cdot (T_{compress} + T_{disk}),$$

where $m = P(\alpha - 1)/S$, among P is the physical memory capacity, S is a single page size (such as 4 KB).

$T_{compress}$: Per-page compression time (approximately 1–5 μs, algorithm-dependent);

T_{disk}: Per-page write latency to storage devices (e.g., HDD: 10 ms, NVMe: 10 μs).

From the computational complexity analysis of these key operations in virtualized systems above, the additional overhead introduced by virtualization can be decomposed into multiple complexity components of different orders, including constant time O(1) and linear time O(n) overhead. For instance, transitions between privilege levels and virtual machine context switches typically incur a fixed overhead of O(1), while operations involving dynamic data processing, such as I/O data transmission and memory page table management, may exhibit O(n) complexity, depending on data scale and access patterns.

In summary, the computational overhead of virtualization results from the combined effects of multiple factors, impacting various dimensions including computation, storage, and networking. Therefore, when designing efficient virtualization systems, it is crucial to consider trade-offs among different types of overhead and adopt optimization strategies to enhance overall system performance while minimizing resource consumption.

C. Isolation Model based on VRSDIM

In intricate blockchain network ecosystems, the primary objective of the VRSDIM is to segregate system resources of gateways, furnishing secure operational environments characterized by divergent security levels. It aims to provide relative isolation for applications spanning across different business domains and network domains, each with distinct security requisites. Consequently, the isolation mechanisms embedded within VRSDIM must adhere to the following constraints:

1: The isolation of operating system kernels entails segregating the kernels across distinct security domains. Each kernel within these domains exhibits relative isolation, featuring autonomously operational processing capabilities, memory allocations, and network interfaces, all within separate runtime environments. Vulnerabilities or malicious code residing within the kernel of a particular security domain are confined solely to that domain, precluding any cross-impact on other domains.

Furthermore, resources among diverse security domains remain inaccessible to each other, thereby effectively mitigating the risk of cross-domain attacks instigated by an untrusted kernel.

2: The isolation among diverse business domains encompasses establishing relative independence in processes, memory access regions, and file storage domains. Each business domain operates autonomously within the security domain, with non-trusted code confined solely to its designated domain. This confinement ensures that any potential malicious activity originating from non-trusted code remains limited to the boundaries of its respective business domain, thereby safeguarding against cross-business domain attacks. Moreover, applications within a given business domain are aligned in terms of security requisites and configurations.

3: The isolation among distinct applications within a shared business domain entails the establishment of relatively autonomous runtime environments, processes, memory allocations, and file storage domains. These elements are segregated to prevent unauthorized cross-access, thereby thwarting unauthorized access attempts and sniffing attacks perpetrated by untrusted applications.

4: The reproducibility of configurable runtime environments is crucial for ensuring the normal operation of programs and maintaining security within computational environments. Particularly, the configuration settings of the file system, operating system, and security functionalities play pivotal roles in program execution and security assurance. Hence, within the VRSDIM framework, the ability to replicate the runtime state of executing programs and the associated system environment is imperative to uphold the availability of security domains. This constraint can be elaborated further as follows:

4a: The runtime environment is governed and managed by a dedicated configuration management domain, which harmonizes configurations across security domains. This entails overseeing the allocation of hardware resources and overseeing security-related aspects, such as password complexity, encryption methodologies, and security protocols.

4b: Security domains lack autonomy in configuring their respective runtime and security environments, encompassing system hardware allocation for system access, file system configurations, and settings pertaining to keys and security protocols within the domain. Instead, security domains are tasked with resource management responsibilities, such as resource invocation and release.

4c: The allocation of root keys within security domains follows a predetermined procedure administered by the management domain, granting write-only permissions. This allocation process assigns root keys to designated storage areas dedicated to individual security domains. Security domains are granted read-only access to their respective storage areas, ensuring the isolation of root keys between domains. Each security domain maintains its distinct root key, inaccessible to root keys associated with other domains. Subsequent keys within a security domain are derived from this root key.

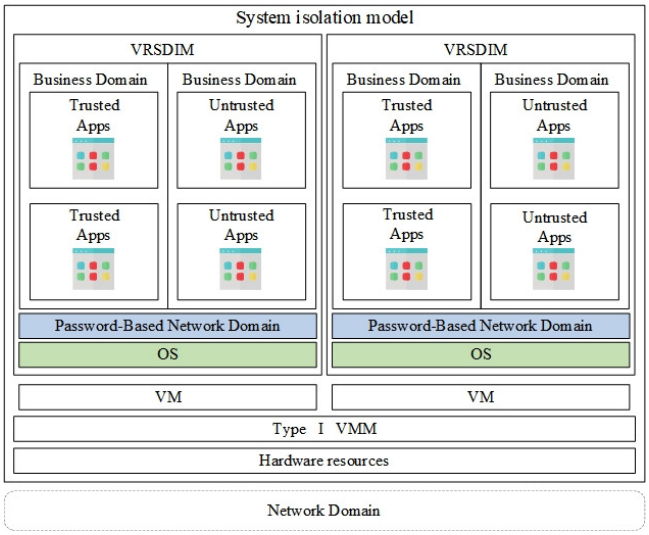

Fig. 3. The isolation model based on the VRSDIM.

5: Prior to initializing the virtual machine monitor (VMM) and the management domain during system boot-up, it is imperative to conduct integrity verification of both the VMM and the kernel to ascertain their integrity.

6: Each security domain is equipped with a password-based network domain, thereby ensuring logical isolation and safeguarding of data within the network.

Fig. 3 depicts the isolation model based on the VRSDIM. In adherence to Constraint 1, VRSDIM necessitates the utilization of the VMM as an intermediary layer for system isolation. The VMM functions by creating replicas of computer systems and software, thereby enabling resource isolation [33]. These replicas, termed virtual machines (VMs), permit the execution of a subset of the processor's instruction set directly on the physical processor. Reference [33] delineates two categories of VMMs: Type I (full virtualization) and Type II (paravirtualization). In Type I, the VMM operates directly on the hardware and manages resource allocation and invocation autonomously. Conversely, Type II functions as an application layer atop the existing host operating system, whereby the actual control over system resources is governed by the host OS. Full virtualization VMMs offer superior isolation capabilities, while paravirtualization VMMs rely on the host operating system for resource allocation, potentially deviating from the model's isolation requirements. Furthermore, full virtualization VMMs necessitate comprehensive hardware driver support, whereas paravirtualization VMMs can leverage hardware drivers directly from the host OS. In terms of implementation, full virtualization VMMs may encounter challenges with hardware compatibility, particularly in complex hardware systems, although this is less of an issue for relatively straightforward gateway systems.

Container and sandbox technologies are capable of fulfilling

the isolation requisites delineated in Constraints 2 and 3 between business domains and applications. Containers facilitate runtime environments and isolation for processes operating within them via virtual resource configuration, ensuring resource segregation across indivitual physical businesses. Sandboxes, on the other hand, confine individual applications within closed environments, minimizing their system-wide impact and guaranteeing isolation between applications. To satisfy Constraint 4, VRSDIM must integrate a management domain tasked with configuring and overseeing the virtual resources and security environments employed by other security domains. Constraint 5 mandates hardware support for assessing the integrity of the loaded system. Meanwhile, Constraint 6 dictates that virtual network interfaces within security domains must offer encryption and configuration capabilities, thereby ensuring logical data isolation between disparate business domains on the physical network.

D. The Security Analysis of VRSDIM

The preceding section delineated the VRSDIM architecture leveraging full virtualization VMM. This section extends the discussion by furnishing a quantitative assessment of VRSDIM's security. To facilitate this evaluation, the subsequent definitions are formulated:

$$\begin{aligned} S &= \{p|\ p\ is\ a\ program\}, \\ ST &= \{p|\ p\ is\ a\ trusted\ program\}, \\ SU &= \{p|\ p\ is\ an\ untrusted\ program\} = S - ST, \\ SM &= \{p|\ p \in SU, and\ contains\ malicious\ code\}, \\ SI &= \{p|\ p \in SU, and\ is\ harmless\} = SU - SM, \\ SV &= \{p|\ p \in ST, and\ contains\ vulnerable\ code\}, \\ SS &= \{p|\ p \in ST, and\ is\ security\} = ST - SV. \end{aligned}$$

VMMs and operating systems (OS) represent distinct categories of software, distinguished by their dual role as both programs and runtime environments indispensable for the execution of other software applications.

$$\begin{aligned} &S_{env} = \{p|\ p\ is\ a\ program\ running\ in\ the\ env\}, \\ &env \in ENV = \{OS, sOS, vOS\}. \end{aligned}$$

"OS" is the conventional operating system, "sOS" is to the operating system for the booting process of the VMM system, and "vOS" is the operating system running on a virtual machine.

$P(p), p \in S_{env}$: The possibility of the program in the env destroying the operating environment of the system.

$P_M(p), p \in S_{env}$: The possibility of the malicious program in the env destroying the operating environment of the system.

$P_V(p), p \in S_{env}$: The possibility of the vulnerable program in the env destroying the operating environment of the system.

$Size(p)$: The quantification of the size of programs, denoting the count of source code lines in the program p.

Drawing from the definition provided earlier, the subsequent formula can be derived:

$$S = ST \cup SU = (SS + SV) \cup (SI + SM). \tag{1}$$

$$P(p) = P_M(p) + P_V(p). \tag{2}$$

As the quantity of programs within the runtime environment env escalates, the security of env diminishes. This deduction can be expressed through the following formula:

$$P(S_{env}^{'}) < P(S_{env}^{''}),\ S_{env}^{'} \subset S_{env}^{''}, \tag{3}$$

where $P(S_{env}^{'}) = \sum_{p \in S_{env}^{'}} P(p)$, $P(S_{env}^{''}) = \sum_{p \in S_{env}^{''}} P(p)$.

$P(S_{env})$ indicates the probability that the security of the operating system in the env is destroyed by programs. $S_{env}^{'}$ and $S_{env}^{''}$ are program collections having different quantities in the env.

Assuming similar code quality, complexity, and implementation, it is estimated that, on average, there exists one Malicious code per one thousand lines of code in software p. The probability of software p containing malicious attacks is proportional to $Size(p)$ so that $P_M(p)$ can be reduced to the following formula:

$$P_M(p) = \alpha \times \frac{Size(p)}{\sum_{p_i \in S_{env}} Size(p_i)},\ p \in S_{env}, \tag{4}$$

where α represents an empirical constant.

Assuming similar code quality, complexity, and implementation, it is estimated that, on average, there exists one security vulnerability per one thousand lines of code in software p. The probability of software p containing vulnerabilities is proportional to $Size(p)$ so that $P_V(p)$ can be reduced to the following formula:

$$P_V(p) = \beta \times \frac{Size(p)}{\sum_{p_i \in S_{env}} Size(p_i)},\ p \in S_{env}, \tag{5}$$

where β represents an empirical constant.

For a conventional multitasking operating system OS, the probability $P(S_{os})$ that the security of the operating system in that env is destroyed by programs can be expressed by the following formula:

$$\begin{aligned} P(S_{OS}) =& P(ST_{OS} \cup SU_{OS}) = P(ST_{OS}) + P(SU_{OS}) \\ =& P(SS_{OS}) + P(SV_{OS}) + P(SI_{OS}) + P(SM_{OS}) \\ =& P(SV_{OS}) + P(SI_{OS}) + P(SM_{OS}). \end{aligned} \tag{6}$$

In equation (6), $P(SI_{OS})$ can be excluded as, according to the definition, the programs within SI_{OS} are deemed devoid of vulnerabilities. With these determinations, this paper proceeds to conduct a security analysis of the VRSDIM architecture. Within the VRSDIM model architecture, the locally initiated OS includes sOS and vOS:

$$P(S_{sOS}) = P(SU_{sOS}) = P(SI_{sOS}) + P(SM_{sOS}). \tag{7}$$

$$\begin{aligned} P(S_{vOS}) &= P(ST_{vOS}) = P(SS_{vOS}) + P(SV_{vOS}) \\ &= P(SV_{vOS}). \end{aligned} \tag{8}$$

For that there are only the VRSDIM VMM, network card device drivers, and components responsible for the operating

system network protocol stack to exchange data within vOS, the formuals can be deduced as follows:

$$S_{vOS} \cong \{VMM, Network\ Components\},$$
$$S_{vOS} \subset S_{OS}\ and\ |S_{vOS}| \ll |S_{OS}|,$$
$$Size(VMM) + Size(Network\ Components) \ll Size(OS). \tag{9}$$

From (3), (4), (5), (8), and (9), we can derive the following inequalities:

$$P(S_{vOS}) \cong P(VMM) + P(Network\ Components) \ll P(S_{OS}). \tag{10}$$

The software size of VRSDIM VMM is significantly smaller compared to traditional operating systems, enhancing the reliability of VRSDIM VMM even when $|S_{VRSDIM}| = |S_{OS}|$. The likelihood of isolated code within the VRSDIM virtual machine compromising the security of vOS can be defined as follows:

$$P(S_{sOS}|VMM|vOS) + P(S_{vOS}) = P(S_{sOS}) \times P(VMM) \times P(vOS) + P(S_{vOS}). \tag{11}$$

In (11), $P(S_{sOS}|VMM|vOS)$ represents the likelihood of the programs within S_{sOS} concurrently breaching the security defenses of sOS, VMM, and vOS. Utilizing (3) and (9), and considering $|\{VMM\}| = |\{vOS\}| = 1 \ll |S_{OS}|$, the following formula can be deduced:

$$P(S_{sOS}|VMM|vOS) + P(S_{vOS}) = P(S_{sOS}) \times P(VMM) \times P(vOS) + P(S_{vOS}) \ll P(S_{OS}). \tag{12}$$

In conclusion, (12) illustrates that VRSDIM effectively bolsters the security of vOS.

IV. System Design

In this section, the system design for CrossGuardian will be introduced from three aspects: (1) Cross-Chain Gateway Architecture; (2) CrossGuardian Isolation Model; and (3) CrossGuardian Isolation Mechanisms.

A. Cross-Chain Gateway Architecture

To support the switching between different cross-chain modes, the cross-chain gateway system is structured into three layers: the application chain layer, the interaction layer, and the relay layer.

Application Chain Layer: This layer comprises the individual blockchain application clients or functional modules, responsible for handling specific business logic and data storage. It provides a unified interface for the interaction layer.

Interaction Layer: This layer is responsible for processing cross-chain transactions according to cross-chain protocols, ensuring effective communication between different blockchains. It includes cross-chain protocols, smart contracts, and middleware, handling the interaction logic between various blockchains. Positioned at the lower level of the cross-chain gateway, the interaction layer contains execution and monitoring modules and relies on the relay layer to implement and coordinate cross-chain operations. The interaction layer abstracts the details of fetching and submitting cross-chain transactions from the application chain layer to the upper-level modules, providing a streamlined interaction interface.

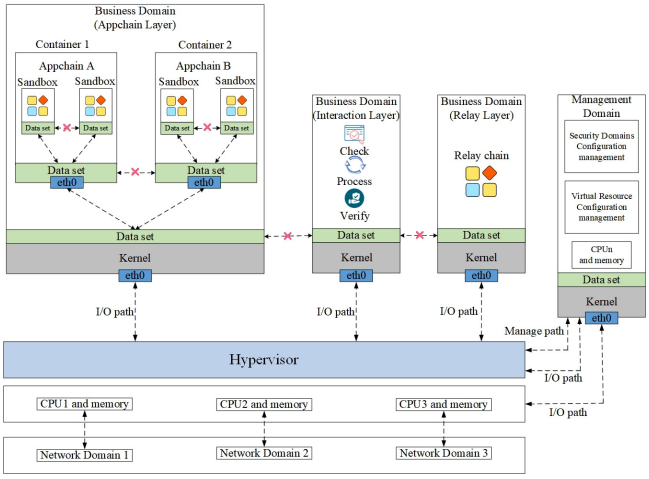

Fig. 4. The architecture of the CrossGuardian.

Relay Layer: This is the core of the cross-chain gateway system, responsible for forwarding and validating cross-chain operations. It monitors and records state changes across different chains, forwards cross-chain transactions, and ensures data integrity and consistency. Additionally, it handles the routing and relaying of cross-chain transactions.

B. CrossGuardian Isolation Model

To ensure a secure operating environment for the cross-chain gateway under complex network and business conditions, our plan CrossGuardian leverages the VRSDIM technology to achieve secure isolation of different layers of cross-chain gateways.

In the VRSDIM isolation mode, the security domain is divided into the management domain, business domains, and network domains. The management domain and business domains operate as virtual machines on the Hypervisor layer. The management domain, a high-privilege virtual machine, is responsible for allocating appropriate hardware resources to the corresponding business domains based on the specific requirements of each functional module. The business domain is tasked with running different functional modules. The network domain provides network interconnection services, ensuring secure communication and data transmission between virtual machines and functional modules.

In the architecture of the CrossGuardian based on the VRSDIM which is shown in Fig. 4, we use business domains to achieve layered isolation for the application chain layer, interaction layer, and relay layer in the cross-chain gateway, in which the data sets in different domains can't be accessed by each other. Multiple appchains in the same appchain domain can be isolated by containers and mutiple application service modules in the same application chain can be isolated by sandboxes, in which the data sets in different appchains and in

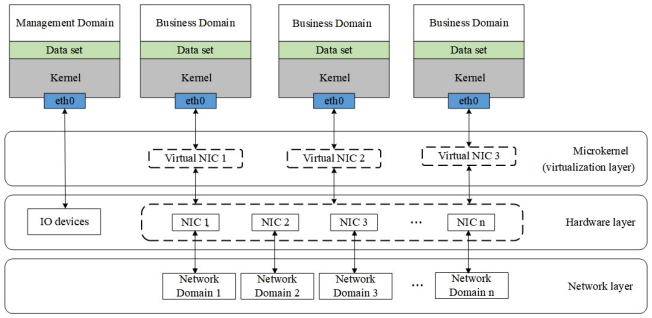

Fig. 5. The configuration and management of network domains.

different service modules can't be accessed by each other. It is efficient to improve the security and stability of the gateway system.

The construction of the business domain and the configuration of system resources are managed by the management domain. The business domain can only access CPU and memory resources through the Hypervisor layer and can only interact with virtual devices via the management domain, without direct access to physical devices. The Management domain is responsible for isolating business domains and allocating hardware resources by creating resource configuration management modules that can operate independently. The resource configuration management module can dynamically manage virtual resources such as processors and memory within the business domain, which ensures that different functional modules in the cross-chain gateway utilize isolated and independent hardware resources while also meeting real-time business needs. Thus, the layered isolation of the CrossGuardian can effectively prevent security degradation, attack propagation, and other security threats arising from plugins or functional modules in different application chains and relay chains.

To ensure the security of network communications between different blockchains within the cross-chain gateway, CrossGuardian uses Hypervisor-based virtual resource isolation to bind multiple physical network interface cards (NICs) to distinct network domains, creating relatively isolated and independent communication channels. The network domains can be configured and scheduled through the configuration management module, ensuring simultaneous or time-sharing multi-domain access capabilities for the cross-chain gateway and establishing a secure network communication environment. The network domain encompasses various connection methods, including WiFi, ZigBee, 3G/4G/5G multimodal communication, Bluetooth, and ultra-shortwave communication. When a business domain attempts to access a network domain, it first initiates a call to the local virtual NIC. The virtual NIC driver then sends the operation request and identity information to the management domain. The management domain, in turn, uses the local device driver to access the corresponding physical NIC, thereby enabling access to the specific network domain.

The configuration and management of network domains in the CrossGuardian isolation model are illustrated in Fig. 5.

The CrossGuardian isolation architecture ensures both control isolation of module instructions and access isolation of data sets within the cross-chain gateway. Cross-chain transactions initiated from the appchain first arrive at the physical NICs in various network domains. These transactions are then passed to the virtualization layer (Hypervisor), which functions as an intermediary, similar to a switch, within the host. The hypervisor is responsible for directing transactions to the business domain within a designated VM, based on the virtual network port configuration. Subsequently, the transactions are bridged to the container's virtualized network port which the appchain plug-in in the sandbox listens on to receive and process the transactions. Once the processing is completed, the results are sent back the hypervisor along the reverse path, ultimately being forwarded to the business domain housing interaction modules by the hypervisor. Notably, direct communication between virtual machines is prohibited, with the hypervisor serving as the intermediary for all inter-VM communication. Finally, the transactions that are converted by the interaction layer are transmitted to the business domain (relay layer) to await verification and be forwarded to the Relay Chain, completing the cross-chain transaction flow. This multi-layered architecture ensures robust isolation and secure, efficient transaction processing across different virtualized environments.

C. CrossGuardian Isolation Mechanisms

In the process of cross-chain transactions, CrossGuardian ensures strict isolation between different layers within the cross-chain gateway, effective data isolation within application processes, and logical isolation of network domains. The cross-chain transaction process based on Crossguardian is depicted in the Fig. 6.

The AppChains (e.g., Chain A, B, C, D as shown in Fig. 6) serve as both the starting and ending points for cross-chain transactions, ensuring the security of their own ledgers and the validity of transactions. When a cross-chain transaction is initiated, the assets being transferred out of the originating chain are locked, while corresponding assets are minted or released on the target chain. Acting as a protocol bridge between heterogeneous chains, the cross-chain gateway is compatible with various chain communication protocols (such as the UTXO model and account model). It monitors the status of cross-chain transactions on the Appchains and converts transaction data into a format that the target chain can recognize. The RelayChain aggregates transaction data from all connected chains, ensures cross-chain transaction finality through its consensus mechanism, and serves as a trust anchor between different blockchains. The specific transaction execution process is as follows:

1. Cross-Chain Event Initiation (AppChain A):

1) A smart contract on AppChain A (e.g., Ethereum simulated via Ganache) triggers a cross-chain transaction request (e.g., data retrieval from AppChain D).

2) The event includes metadata such as source/destination chain IDs, transaction parameters, and cryptographic signatures.

2. Event Monitoring (AppChain Domain):

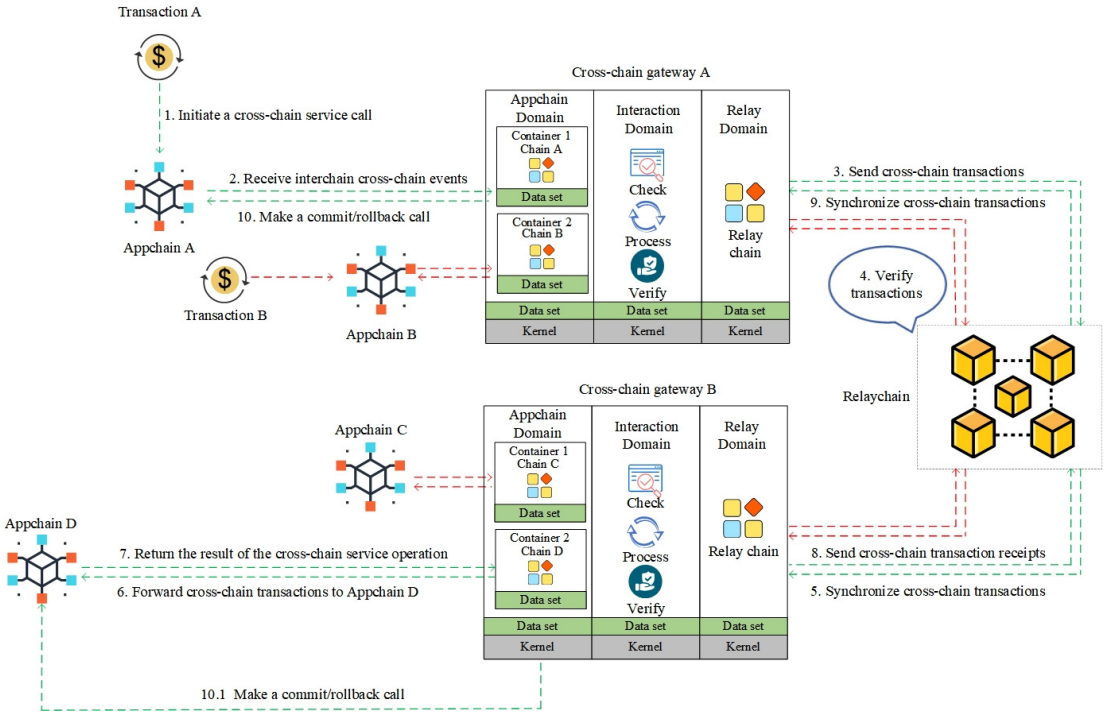

Fig. 6. The cross-chain transaction process based on Crossguardian.

1) AppChain A Plugin within the AppChain Domain (isolated via containers) continuously listens for cross-chain events.

2) The plugin acts as a lightweight client, ensuring protocol-specific communication with AppChain A while maintaining isolation from other chains.

3. Transaction Forwarding (Interaction Domain):

1) The transaction is routed to the Interaction Domain, a dedicated security layer responsible for protocol conversion (e.g., from native AppChain format to IBTP).

2) Data validation (e.g., nonce checks, signature verification) is performed here to filter malicious or malformed requests.

4. Relay Chain Submission (Relay Domain):

1) Validated transactions are encapsulated into relay chain-compatible formats (e.g., BitXHub's IBTP) and forwarded to the Relay Domain.

2) The RelayChain plug submits the transaction to IPFS (simulated relay chain), ensuring persistence and consensus-driven state synchronization.

5. Cross-Chain Propagation (Relay Chain):

The relay chain processes the transaction through consensus (e.g., Raft/PBFT), verifies its validity, and routes it to the target gateway (CrossChain Gateway B).

6. Target Chain Execution (AppChain Domain):

1) CrossChain Gateway B receives the transaction via its Relay Domain, parses the IBTP payload, and converts it into a format compatible with AppChain D (e.g., Fabric-specific transactions).

2) The transaction is executed on AppChain D through its isolated plugin, with results recorded on-chain.

7. Receipt Generation (Interaction Domain):

Post-execution, a cryptographically signed receipt is generated by AppChain D's plugin and forwarded to the Interaction Domain for format standardization.

8. Receipt Propagation (Relay Chain):

The receipt undergoes the same relay chain validation and routing process (Steps 4–5), ensuring atomicity and consistency across chains.

9. Source Chain Synchronization (AppChain Domain):

The original gateway (CrossChain Gateway A) receives the receipt via its Relay Domain, verifies its integrity, and updates the state on AppChain A.

10. Commit/Rollback (Smart Contract):

AppChain A's smart contract finalizes the transaction (commit) or triggers a rollback based on receipt validity, ensuring cross-chain atomicity.

Fig. 6 encapsulates CrossGuardian's core innovation: layered isolation without sacrificing interoperability. By decomposing cross-chain workflows into domain-specific, hypervisor-mediated steps, the model ensures that vulnerabilities in one layer (e.g., a compromised AppChain plugin) cannot propagate to others (e.g., relay or interaction domains). The figure also underscores the role of standardized protocols (e.g., IBTP) in enabling secure, heterogeneous blockchain interoperability.

The security threats that the cross-chain gateway may encounter in the process of processing cross-chain transactions, such as unauthorized access, untrusted firmware upgrades, cross-chain attack propagation, and data leakage can be effectively solved by the CrossGuardian security isolation model proposed in this paper, which is as the Fig. 7.

1. Unauthorized Access Prevention: This issue arises from the lack of authentication and authorization mechanisms

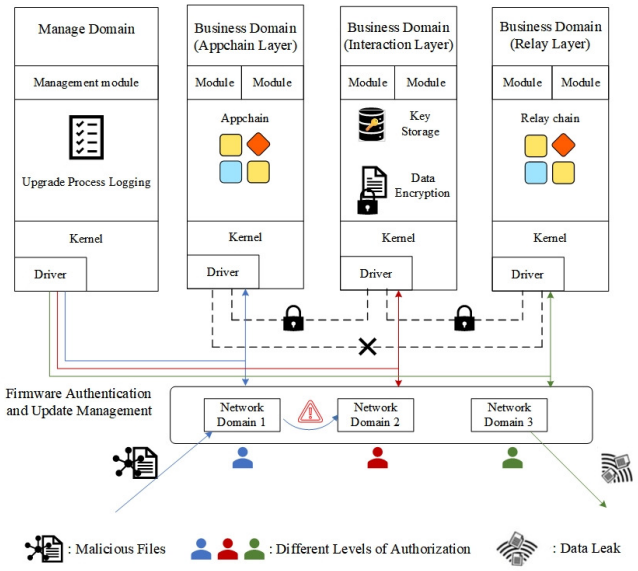

Fig. 7. The security protection against threats in the CrossGuardian.

within the network domain. Our model can efficiently address this through authentication technologies implemented in the logically isolated network domain, such as role-based access control (RBAC) and attribute-based access control (ABAC) [34]. Additionally, the business domains can only have access to their own data set and communicate confidentially with each other through the hypervisor, rather than directly accessing one another, effectively preventing unauthorized access.

2. Protection Against Untrusted Firmware Upgrades: The vulnerability stems from the lack of authentication for the firmware source and the process running the upgrade, as well as potential extensive interactions and improper permission management among components of the cross-chain gateway. To address this, the security domain can authenticate and safeguard the firmware source and the environment in which the upgrade process runs. Additionally, this model can restrict firmware upgrade permissions, allowing upgrades only through the management domain while recording the upgrade process for traceability and auditing purposes.

3. Prevention of Cross-Chain Attack Propagation: The risk arises from traditional cross-chain gateways where different blockchain plugins share software and hardware resources. To address this, our model securely isolates the layers in the cross-chain gateway by business domains. By routing interactions through the Appchain domain → Interaction domain (a specialized domain) → Relay domain, it effectively prevents the propagation of attacks between chains.

4. Data Leakage Prevention: The risk stems from insecure data transmission and storage, weak password management, and insufficient access control. To address this, our model can flexibly employ configurable encryption algorithms within the isolated Interaction domain to encrypt sensitive data. Additionally, the keys used for cross-chain communication between multiple blockchains are stored within the Interaction domain, which does not communicate with external entities, significantly reducing the risk of key leakage and ensuring data security during storage and transmission. Furthermore, the Interaction domain restricts access to sensitive data, allowing only authorized users or systems to access it, with access activities being recorded for monitoring and auditing purposes.

Our model layers different functional modules which can invoke fewer hardware resources than before so that in the special environments our proposed model CrossGuardian is vulnerable to the security threats, such as high concurrency and distributed denial of service (DDoS) attacks. The business domains can congestion and downtime under the high load of cross-chain business. Buffer pools can mitigate contention and prevent system overloads under high-concurrency workloads. Complementing this, load balancing strategies distribute incoming requests across a pool of domains or different modules, ensuring an equitable utilization of resources. A DDoS-aware model is configured within the management domain, and upon detecting a DDoS threat, the system automatically takes action—ranging from reconfiguring resources to shutting down the attacked domain and enabling backup domains to continue providing services [35]. Additionally, particularly those with optimistic concurrency control, the system in the isolation model may mistakenly detect conflicts or inconsistencies that do not exist. This can result in unnecessary rollbacks, wasting computational resources and introducing delays. Transaction Window Tightening narrows the scope and duration of a transaction, which can efficiently reduce overlap and minimize unnecessary conflicts. The strategy of Transaction Prioritization assigns priority levels to transactions, enabling high-priority transactions to proceed with fewer conflict checks, while lower-priority transactions may be delayed, which also helps reduce contention and minimize rollbacks.

V. Experiments

In this section, we implemented a secure operating environment for the cross-chain gateway system using the model proposed in this paper. We simulated the process of forwarding, processing, and recording cross-chain transactions across different business domains within the isolated environment. Additionally, we conducted tests and evaluations on both the business domain's system performance and communication performance.

A. System Configuration and Architecture

The system architecture of the cross-chain gateway in the simulation experiment was conducted on an X86 architecture PC platform. The testing platform utilized a 64-bit Intel i3-10100 processor, which features 8 physical CPU cores at 3.60 GHz, 6 MB of L3 cache, and a thermal design power (TDP) of 65 W. Intel's Hyper-Threading technology was disabled on the testing platform to provide a hardware environment as similar as possible to platforms that do not support this feature, such as ARM platforms, for comparative purposes. The testing platform is equipped with 8 GB of RAM, 1 TB of solid-state storage, and a USB 3.0 1000M/b

TABLE II
SYSTEM CONFIGURATION OF THE TEST PLATFORM.

Processor	Intel I3-10100, 8 cores, 3.6 GHz, 6MB L3 cache, TDP 65 W.
Memory	8 GBDDR4
Storage	1 TB SSD
Network card	USB3.0 interface 1000 M/b Ethernet card.
Operating system	Ubuntu Core 20.04
Hypervisor	Xen 4.11.0
Kernel	Linux 5.15.0

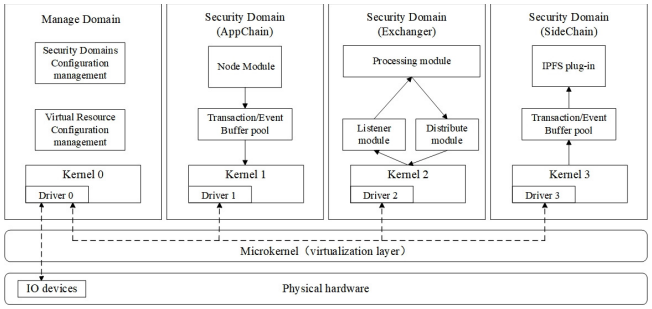

Fig. 8. The architecture of cross-chain gateway in the experiment.

Ethernet card. The system configuration of the testing platform is detailed in Table 2.

The cross-chain gateway testing system utilizes the Xen virtualization framework version 4.11.0 on an x86 architecture. The business domains (DomU) are configured to use hardware virtual machine (HVM) full virtualization, whereas the management domain (Dom0) supports only PV instances. All hosts and VMs run Ubuntu 20.04, with the same Linux 5.15.0 kernel and identical software configurations across all machines. The testing system comprises 3 business domains (DomU) and 1 management domain (Dom0). Each business domain (DomU) is allocated 2 processor cores, 2 GB of memory, and 10 GB of storage. The management domain (Dom0) is allocated 2 processor cores, 1 GB of memory, and the remaining system storage. The management domain is connected to the test server via a 1000 Mb/s physical network card and is bound through a dedicated network bridge to provide network services to virtual network cards within the business domains.

Based on the description in Section 4.2, in the Cross-Guardian isolation model, the functional modules of the cross-chain gateway are isolated by business domains (DomU). The management domain is configured to ensure that the hardware resources for each domain meet normal computational requirements and that network communication between domains are functioning properly. To validate the effectiveness of VRSDIM in practical blockchain gateway applications, in the experiment we used Ganache, the interplanetary file system (IPFS), and the Interaction plug-ins to simulate the entire process of cross-chain transaction reception, processing, and forwarding. The architecture of cross-chain gateway in this experiment is illustrated in the Fig. 8.

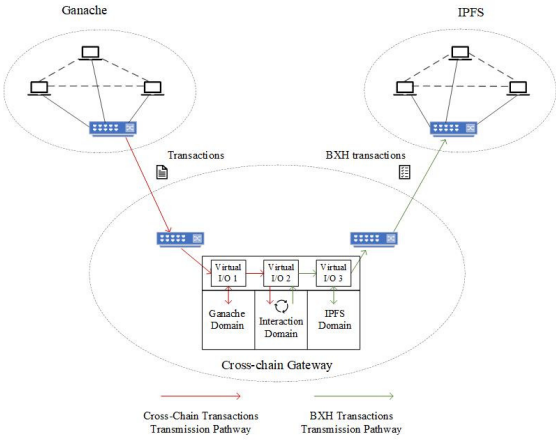

Fig. 9. The design of the cross-chain experimental network topology.

B. The Design of Cross-Chain Transaction Experiment

Ganache, as a test chain node for Ethereum, was used to simulate the generation and processing of blockchain transactions. IPFS, a fully decentralized, content-addressed media object storage and retrieval platform, was employed to simulate a relay chain for storing and sharing the cross-chain transaction. The Interaction domain, as the core module for transaction processing and forwarding, is responsible for the conversion and transmission of transactions between different blockchains. The cross-chain experimental network topolog is illustrated in the Fig. 9:

The experiment simulated specific cross-chain gateway processing cross-chain transactions, such as the conversion of data format. The cross-chain gateway converts cross-chain transaction formats into a structure commonly used by the relay chain. For instance, using the BITHUB cross-chain protocol (IBTP), the experiment simulated a Ganache node generating and listening for a cross-chain transaction, then sending the relevant transaction data to the Interaction domain. The Interaction domain receives the transaction request and converts the cross-chain transaction format into the IBTP structure that can verified by the relay chain. Subsequently, the IBTP structure is encapsulated into the bxh transaction type and submitted to IPFS for storage and sharing.

The format of the cross-chain transaction is as follow.

```
{
    "transaction": {
    "id": "tx1234567890"
    "timestamp": "2024-08-11T14:30:00Z",
    "source_chain": "ChainA",
    "destination_chain": "ChainB",
    "anount": 1000,
    "currency": "TOKEN",
    "sender": "0xSenderAddress",
    "receiver": "0xReceiverAddress",
    "data": {
        "additional_info": "This is an example of a
↪  cross-chain transaction."
    }
  }
}
```

The format of the converted bxh transaction is as follow.

```
{
  "bxhTx1": {
// Adress of the cross-chain gateway
```

```
    "From":"0xCrossChainGatewayAddress",
// Adress of the BitXHub contract handing
↪  cross-chain transactions
    "To":"0xBitXHubContractAddress",
    "IBTP": {
      "header": {
        "transaction id": "tx1234567890",
        "timestamp": "2024-08-11T14:30:00Z",
        "source_chain": "ChainA",
        "destination_chain": "ChainB",
        "currency": "TOKEN",
        "metadata": {
          "transaction_type": "cross_chain",
          "status": "pending",
          "priority": "high"
        }
      },
      "body": {
        "amount": 1000,
        "sender": "0xSenderAddress",
        "receiver": "0xReceiverAddress",
        "data":{
          "additional_info": "This is an example of
↪  a cross-chain transaction."
        }
      }
    },
// Represents the index of the IBTP, usually
↪  generated by the system
    "Nonce": 1,
// Timestamp of the cross-chain event
    "Timestamp": "2024-08-11T14:30:00Z"
// Additional parameters can be added as needed
  }
}
```

The analysis of the experimental results indicates that the CrossGuardian isolation model successfully enables the normal operation of the fundamental functions of the cross-chain gateway, ensuring interaction between different blockchains and the forwarding of cross-chain messages. Additionally, it effectively prevents threats such as security degradation and attack propagation caused by the mixing of data from different types and security levels.

C. Security Testing Experiment

To validate the robustness and effectiveness of our proposed CrossGuardian isolation model under various attack scenarios, we conducted two simulation experiments. These simulations were designed to address potential challenges in real-world network environments, including single-domain interruptions caused by attacks or module failures, as well as the propagation of malicious threat data across multiple domains.

Experiment 1: Single-Domain Interruption Simulation

In this experiment, we focused on simulating an attack scenario where a single domain within the network is interrupted due to external threats or internal component failures. The primary objectives were to assess the system's ability to isolate compromised domains and restore functionality under such conditions.

1. Attack Scenario: An adversary launched a denial-of-service (DoS) attack on the business domain (AppChain), causing temporary communication outage of the Application Chain Layer from other layers in the cross-chain gateway.

2. Defense Mechanisms: The management domain (Dom0) detects anomalous cpu and memory of the Dom1 through real-time monitoring (the status shown in Fig. 10), triggers

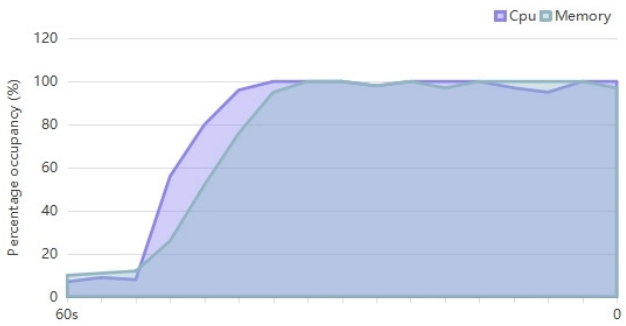

Fig. 10. Dom1 Real-time performance monitoring (DOS).

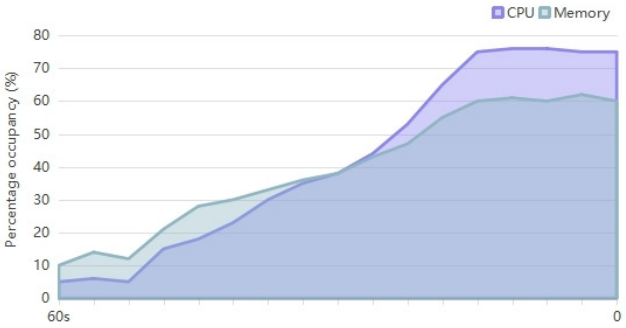

Fig. 11. Dom1 Real-time performance monitoring (APT).

the backup domain (a pre-configured redundant AppChain domain) to start.

During the attack, interaction Domain and Relay Domain sevices were not affected, throughput decreased by 8% (from 320 TPS to 295 TPS) and latency increased by 15% (0.19 s $\rightarrow$ 0.22 s on average). After activating the backup domain, the AppChain plugin within the AppChain domain (isolated via containers) and the interaction domain resended the previously failed messages due to interruptions, in order to restore communication.

Experiment 2: Cross-Domain Threat Propagation Simulation

This experiment aimed to evaluate the CrossGuardian isolation model's ability to handle malicious threat propagation across multiple domains, simulating an advanced persistent threat (APT) scenario.

1. Threat Propagation: An attacker initiated an APT by compromising one node in Domain 1. The threat propagated through Domain 1, then spread to Domains 2 and 3 via shared common communication bus.

2. Defense Mechanisms: The management domain (Dom0) detects anomalous cpu and memory of the Dom1 through real-time monitoring (the status shown in Fig. 11) and immediately isolates the virtual network interface of Dom1. Cross-domain communication is restricted to allow only whitelisted protocols such as IBTP, preventing the spread of malicious packets.

In the simulation, the Domain 1 CPU usage peaked at 75% (15% normal load) and the memory footprint increased by 1.2 GB. During the attack, Dom2 and Dom3 saw a 9% increase

TABLE III
BENCHMARK TEST PROGRAMS.

Test programs	Parameter settings
Hackbench	Using system Sockets, configure 100 process groups (parameter g set to 100),each group utilizing 40 file descriptors (equivalent to 4000 CPU tasks), with each file descriptor handling 500 messages containing 100 bytes each (parameter l set to 500).
OpenSSL	Use the OpenSSL 1.1.1 "speed" testing tool to assess the performance of SHA-256, AES-128, AES-256, RSA 2048, RSA 4096 and ECDSA algorithms on the target system with default settings.
Netperf	Start the netserver service on the server side and use Netperf v2.6.0 on the target platform to perform latency and throughput performance tests on the target platform's network using TCP_RR, TCP_STREAM, and TCP_MAERTS test modes with default parameter settings.
Caliper	Generate a workload targeting a specific system under test (SUT) and continuously monitor its corresponding responses. Based on the observed responses of the SUT, generate a report detailing the findings.

in cross-chain transaction latency (0.12 s → 0.13 s) and no significant decrease in throughput (maintaining 305 TPS).

Experimental results demonstrate that CrossGuardian effectively addresses security challenges in network defense. In terms of single-domain outages, the system successfully isolates and mitigates attacks within a timely manner, restoring service within 5 minutes to meet high availability requirements. Furthermore, in cross-domain propagation scenarios, APT attacks are strictly contained to the initial infection domain without lateral spreading, thereby validating the efficacy of virtual network isolation techniques combined with protocol whitelisting.

D. Performance Testing Experiment

Based on the analysis of the simulation experiment, compared to traditional gateway architectures, the performance of CrossGuardian in handling cross-chain transactions is primarily influenced by security isolation and inter-domain communication. To validate the system and communication performance of the business domains within the virtualization technology framework, this study employs four benchmark testing programs: Caliper, Hackbench, OpenSSL, and Netperf. These tools are used to test the performance of the management and business domains in operating blockchain transactions, process scheduling, common encryption algorithms, and network communications. To minimize the impact of different software versions on system performance testing, the experiments utilize the latest versions of the benchmark programs as much as possible. The testing programs are listed in Table 3.

Experiment 1: Processing Performance Testing

To test the performance of the business domain in handling blockchain transactions, we first set up a Fabric test chain within the business domain and used Caliper to conduct performance testing on the test network through sample components.

In this simulation test, we evaluated the performance of the business domain in the Hyperledger Fabric v2.2 test chain under various operational scenarios, including data creation,

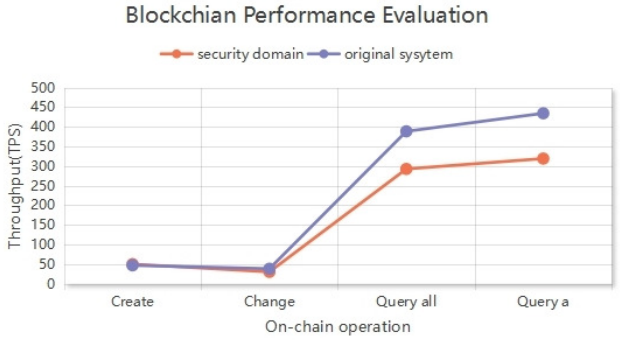

Fig. 12. The comparison of blockchain throughput in different systems.

modification, full query, and single query. Fabric v2.2 primarily uses the Raft consensus algorithm, a log-based consensus protocol that provides an efficient consensus mechanism suitable for most commercial blockchain applications. The chain employs the following cryptographic algorithms: symmetric encryption with the advanced encryption standard (AES) algorithm; asymmetric encryption with the elliptic curve digital signature algorithm (ECDSA) for digital signatures and key exchange; and hashing with secure hash algorithm (SHA-256) for data hashing. As presented in the Table 4, the test simulated real-world on-chain operations to assess the isolation system's throughput, latency, and success rate when handling different types of transactions. The Fig. 12 compares the throughput performance of the business domain with the original system in processing blockchain transactions. According to the experimental results, there is a negligible gap between the CrossGuardian isolation system and the original system in the processing of blockchain transactions. However, due to the original system has more hardware resources and bandwidth than the security domain, the TPS difference increases when the system processes the querying blockchain transactions that has a higher number.

To evaluate the impact of security domain isolation model on data processing performance, we employed two benchmark applications, Hackbench and OpenSSL, to test the processor's multitasking and multiprocess handling capabilities, as well as the performance of SHA, AES, RSA, and ECDSA cryptographic algorithms within the management domain, the business domain, and the original system. The primary function of Hackbench is to generate a specified number of schedulable entity pairs (threads or traditional processes) that communicate via sockets or pipes, and calculate the time it takes for each entity pair to send data back and forth. The basic principle of the Openssl speed test is to invoke the designated cryptographic algorithm within a specified period of time (e.g., 3 s or 10 s). The last number of operations represents the performance of the algorithm. We normalized each result of the indicators to facilitate the analysis of experimental data and eliminate dimensional influences between indicators.

In the Hackbench test, 8 processor cores in the original system handled 4000 tasks simultaneously, whereas only 2 processor cores in the management domain and the business domain were used to handle the same number of tasks. The

TABLE IV
BLOCKCHAIN TRANSACTION SIMULATION TESTING.

Name	Succ	Fail	Send rate (TPS)	Max latency (s)	Min latency (s)	Avg latency(s)	Throughput (TPS)
Create a car	5000	0	50.2	1.18	0.03	0.12	50.2
Change car owner	952	0	32.6	2.11	0.03	0.19	30.5
Query all cars	8503	0	292.7	0.06	0.00	0.02	292.6
Query a car	9272	0	319.1	0.15	0.00	0.02	319.0

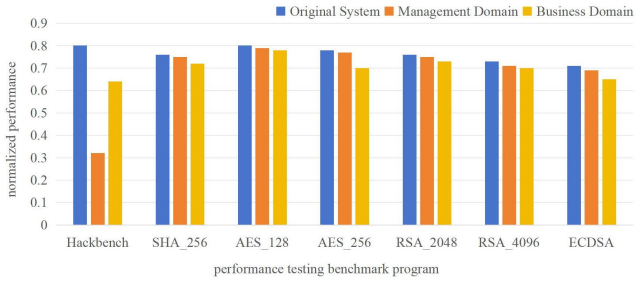

Fig. 13. Processing performance comparison in different systems.

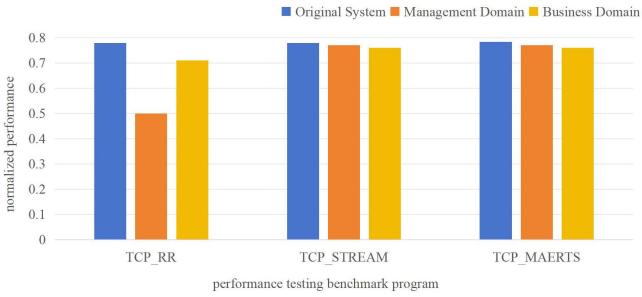

Fig. 14. Communication performance comparison in different systems.

experimental results in Fig. 13 indicate that when performing parallel processing tasks, the number of processors and the amount of allocated memory directly impact the system's performance. The management domain was allocated 2 processor cores and 1 GB of memory, while the business domain had 2 processor cores and 2 GB of memory. If the memory is insufficient for the CPU scheduling requirements, it can negatively affect performance (with the management domain performing worse than the business domain). Conversely, the difference in performance is more proportional to the number of processors (with the original system outperforming the business domain), and less affected by differences in memory size.

According to test results of the Openssl speed, the isolation mechanism has a slight impact on the performance of the system executing encryption algorithms. The main reason is that the business domain cannot directly access the physical hardware. All physical hardware access must go through the management domain and Hypervisor layer, and the management domain can directly access the physical hardware. This access mechanism can result in some performance penalties, but it ensures the secure isolation of system resources.

Experiment 2: Communication Performance Testing

TCP/IP serves as the foundational protocol for data transmission between blockchain nodes, managing packet segmentation, transmission, reassembly, and error checking. It ensures that nodes within the blockchain network can establish stable connections and communicate effectively. Therefore, we employed the Netperf benchmark testing program to evaluate the TCP/IP network performance of both the management domain and the business domain. The Netperf test results specifically reflect the throughput of data transfer between systems in unit time, typically expressed in the amount of data transferred per second (such as bits/s or bytes/s). The test results are shown in the Fig. 14.

TCP_RR test evaluates the performance of handling a large number of TCP requests and replies, while TCP_STREAM primarily measures the performance of sending fixed-size data packets over a TCP/IP network, such as sending a file via FTP. On the other hand, TCP_MAERTS reflects the performance of receiving fixed-size data packets, such as downloading a firmware file. Experimental results indicate that by reasonably allocating network resources through virtualization technology, a security domain can fully meet the data forwarding and processing requirements based on the TCP/IP network.

The cross-chain gateway typically consists of multiple functional components, including common ones like the light blockchain node modules and the cross-chain transaction interaction module. As described in Section 4.2, the different modules within the CrossGuardian model are securely isolated across various business domains, with Inter-Process Communication (IPC) isolation implemented between them. These modules require network communication to connect with each other. To test the communication performance between different business domains, the experiment utilized the Netperf benchmark to assess the performance for processing different sizes of blockchain communication data. The tests included scenarios such as simple asset transfers (data size: 200 to 500 bytes), smart contract invocations (data size: 1 KB to 5 KB), and transactions containing state proofs (data size: 5 KB to 20 KB). The experimental results are shown in the following Fig. 15:

The experimental results indicate that even with security isolation in place, the interaction and transmission of transaction data between different business domains still exhibit good performance. This is primarily because the business domains are isolated on the same host and share a common communication bus, which helps maintain low network latency

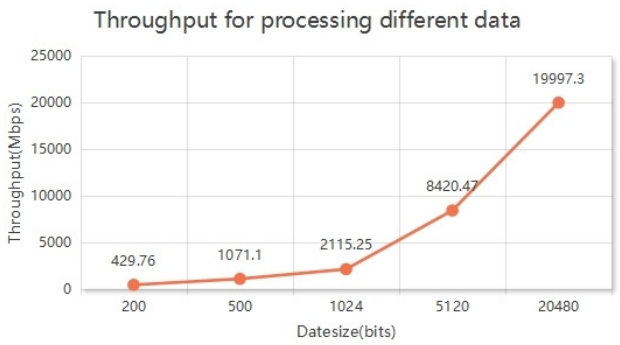

Fig. 15. Inter-domain communication performance for different data sizes.

and jitter, resulting in high throughput.

E. Summary of Experimental Evaluation

The experimental evaluation validates the feasibility and effectiveness of the proposed CrossGuardian model across three key dimensions: system functionality, processing performance, and communication efficiency.

1. System Functionality:

1) CrossGuardian successfully simulated cross-chain transactions using Ganache (Ethereum test chain) and IPFS (relay chain), demonstrating seamless interaction between heterogeneous blockchains. The model effectively converted cross-chain transaction formats (e.g., IBTP protocol) and ensured secure forwarding through isolated business domains.

2) The simulation results of single-domain interruption simulation and cross-domain threat propagation simulation verify that the isolation mechanism is effective in limiting security vulnerabilities to a single domain. By isolating the distinct functional layers of traditional cross-chain gateways within relatively independent execution environments, the CrossGuardian model effectively ensures the separation of data with varying types and security levels. This approach significantly mitigates the risk of security threats (e.g. attack diffusion, unauthorized access).

2. Processing Performance:

1) Blockchain Transactions: According to the Caliper benchmark results on Hyperledger Fabric v2.2, although the TPS of the CrossGuardian isolation system is lower compared to the original system in querying blockchain transactions, the isolation system's throughput meets the basic requirements of the blockchain and enhances the security of cross-chain transactions. Therefore, the performance trade-off is acceptable.

2) Cryptographic Operations: OpenSSL tests revealed minor performance trade-offs (e.g., SHA-256, AES-128) due to virtualization overhead, but results remained within acceptable thresholds for real-world blockchain applications.

3) Multitasking: Hackbench demonstrated efficient parallel task handling (4000 tasks) and performance scaling proportional to allocated CPU cores. Resource allocation strategies (e.g., binding domains to specific cores) are key to balancing security and performance.

3. Communication Efficiency:

Netperf tests confirmed that the CrossGuardian model's layered isolation mechanism for cross-chain gateways does not affect the system's TCP/IP network communication and data transfer between modules. Even for large transactions (20 KB state proofs), inter-domain transmission maintained high efficiency. The primary reason for this is that inter-domain communication between functional modules utilizes a shared communication bus within the host. Network isolation via virtual NICs and hypervisor-mediated routing ensured secure data transmission without significant overhead.

CrossGuardian achieves a balance between security and performance. Although virtualization introduces marginal latency (for example, throughput is reduced by 10–15% compared to non-isolated systems), the enhanced security guarantees make this trade-off justifiable. At the same time, the architecture is adaptable to multi-core hardware, and future optimizations (such as those on ARM chips with a big.LITTLE core architecture, like the domestic RK3399 chip from Rockchip) are expected to further improve performance. This chip features six processor cores: two Cortex-A72 big cores and four Cortex-A53 little cores. The management domain, in addition to being responsible for configuring and managing the resource allocation and state monitoring of the business domains, also handles the access requests to hardware resources from the business domains. These tasks are diverse and complex, while applications within the business domain tend to be relatively fixed and simpler. Therefore, the management domain is better suited to be assigned to a big core, such as the Cortex-A72, while the business domain is more suitable for being assigned to a small core, such as the Cortex-A53. According to Moore's Law, future hardware development will shift from simply increasing processor frequency to increasing the number of processor cores to improve performance. The proposed architecture in this paper will also expand its application range as the number of processor cores increases in the future.

Overall, the experiments substantiate CrossGuardian's viability as a secure, efficient cross-chain gateway framework, addressing critical interoperability challenges while maintaining robust isolation across business and network domains.

VI. Conclusion

This paper introduces a security domain isolation model termed VRSDIM, which is grounded in hardware resource virtualization principles. It further proposes a cross-chain security gateway system architecture predicated on this model. The architecture ensures isolation between different business domains and network domains during cross-chain gateway operations, furnishing a robust underpinning for dependable data aggregation, cross-chain application enhancement, and secure interfacing with diverse blockchain networks. Additionally, this paper entails the implementation of a secure gateway operational environment leveraging the x86 hardware platform and virtualization technology. It simulates the processing cross-chain transactions int the cross-chain gateway and conducts performance evaluations encompassing virtual resource process scheduling, security services, and networking, thereby validating the architectural feasibility and secu-

rity. By surmounting the limitations inherent in conventional cross-chain gateway system architectures, such as hurdles in achieving efficacious transaction information isolation, cross-chain verification information isolation, and cross-chain bridge information isolation, this architecture presents a pioneering framework for cross-chain networks. It offers invaluable insights for research endeavors pertaining to blockchain and cross-chain network security system architectures.

REFERENCES

[1] J. A. Jaoude and R. G. Saade, "Blockchain applications–usage in different domains," *IEEE Access*, vol. 7, pp. 45360–45381, 2019.

[2] L. Y. Zhong, L. J. Wei, Z. Z. Yang, X. T. Ge, and Y. Hui, "Research on consensus mechanisms ofblockchain technology," *J. Cryptologic Research*, vol. 6, no. 4, pp. 395–432, 2019.

[3] G. Caldarelli, "Overview of blockchain oracle research," *Future Internet*, vol. 14, no. 6, p. 175, 2022.

[4] R. Belchior, A. Vasconcelos, S. Guerreiro, and M. Correia, "A survey on blockchain interoperability: Past, present, and future trends," *ACM Computing Surveys (CSUR)*, vol. 54, no. 8, pp. 1–41, 2021.

[5] T. Xie, K. Gai, L. Zhu, Y. Guo, and K.-K. R. Choo, "Cross-chain-based trustworthy node identity governance in Internet of things," *IEEE Internet Things J.*, 2023.

[6] M. Wang, J. Liu, J. Chen, J. Mao, and K. Mao, "Software defined networking: Security model, threats and mechanism," *J. Software*, vol. 27, no. 4, pp. 969–992, 2016.

[7] J. Wang, J. Cheng, Y. Yuan, H. Li, and V. S. Sheng, "A survey on privacy protection of cross-chain," in *Proc. ICAIS*, 2022.

[8] P. Han, Z. Yan, W. Ding, S. Fei, and Z. Wan, "A survey on cross-chain technologies," *Distributed Ledger Technologies: Research and Practice*, vol. 2, no. 2, pp. 1–30, 2023.

[9] M. Borkowski, M. Sigwart, P. Frauenthaler, T. Hukkinen, and S. Schulte, "Dextt: Deterministic cross-blockchain token transfers," *IEEE Access*, vol. 7, pp. 111030–111042, 2019.

[10] S. Yang, H. Wang, W. Li, W. Liu, and X. Fu, "CVEM: A cross-chain value exchange mechanism," in *Proc. CCIOT*, 2018.

[11] S. K. Mohanty and S. Tripathy, "n-HTLC: Neo hashed time-lock commitment to defend against wormhole attack in payment channel networks," *Comput. Security*, vol. 106, p. 102291, 2021.

[12] A. Garoffolo, D. Kaidalov, and R. Oliynykov, "Zendoo: A zk-SNARK verifiable cross-chain transfer protocol enabling decoupled and decentralized sidechains," in *Proc. IEEE ICDCS*, 2020.

[13] W. Hao, S. Xiangfu, K. Junming, and X. Qiuliang, "Blockchain in digital currency and its privacy protection mechanism [j]," *Inf. Netw. Security*, vol. 7, pp. 32–39, 2017.

[14] Y. Han, C. Wang, H. Wang, Y. Yang, and X. Wang, "A study of blockchain-based liquidity cross-chain model," *Plos one*, vol. 19, no. 6, p. e0302145, 2024.

[15] G. Wood, "Polkadot: Vision for a heterogeneous multi-chain framework," *White paper*, vol. 21, no. 2327, p. 4662, 2016.

[16] H. Wang *et al.*, "An electricity cross-chain platform based on sidechain relay," in *Proc. JPCS*, 2020.

[17] S. Zhang, R. Zhou, L. Wang, S. Xu, and W. Shao, "Cross-chain asset transaction method based on ring signature for identity privacy protection," *Electronics*, vol. 12, no. 24, p. 5010, 2023.

[18] X. Liu, Z. Zhou, W. Diao, Z. Li, and K. Zhang, "When good becomes evil: Keystroke inference with smartwatch," in *Proc. ACM SIGSAC CCS*, 2015.

[19] K. Sai and D. Tipper, "Disincentivizing double spend attacks across interoperable blockchains," in *Proc. IEEE TPS-ISA*, 2019.

[20] J. Flood and A. McCullagh, "Blockchain's future: Can the decentralized blockchain community succeed in creating standards?" *Knowl. Eng. Rev.*, vol. 35, p. e2, 2020.

[21] A. Singh *et al.*, "Sidechain technologies in blockchain networks: An examination and state-of-the-art review," *J. Netw. Comput. Appl.*, vol. 149, p. 102471, 2020.

[22] W. Ou *et al.*, "An overview on cross-chain: Mechanism, platforms, challenges and advances," *Comput. Netw.*, vol. 218, p. 109378, 2022.

[23] N. S. Mtetwa, P. Tarwireyi, A. M. Abu-Mahfouz, and M. O. Adigun, "Secure firmware updates in the Internet of things: A survey," in *Proc. IEEE IMITEC*, 2019.

[24] L. Duan *et al.*, "Attacks against cross-chain systems and defense approaches: A contemporary survey," *IEEE/CAA J. Automatica Sinica*, vol. 10, no. 8, pp. 1647–1667, 2023.

[25] S. Rizvi, R. Orr, A. Cox, P. Ashokkumar, and M. R. Rizvi, "Identifying the attack surface for IoT network," *Internet Things*, vol. 9, p. 100162, 2020.

[26] R. Shu *et al.*, "A study of security isolation techniques," *ACM Computing Surveys (CSUR)*, vol. 49, no. 3, pp. 1–37, 2016.

[27] Z. Bing-Nan, D. Liang, and Z. Qing-Kai, "Kernel-level multi-domain isolation model based on hardware virtualization," *J. Software*, vol. 33, no. 2, pp. 473–497, 2021.

[28] H. Lefeuvre *et al.*, "Flexos: Making os isolation flexible," in *Proc. ACM HotOS*, 2021.

[29] D. Schrammel *et al.*, "Donky: Domain keys–efficient {In-Process} isolation for {RISC-V} and x86," in *Proc. USENIX Security Symposium*, 2020.

[30] Y. Xu, C. Ye, Y. Solihin, and X. Shen, "Hardware-based domain virtualization for intra-process isolation of persistent memory objects," in *Proc. ACM/IEEE ISCA*, 2020.

[31] J. M. Kizza and J. M. Kizza, "Virtualization technology and security," *Guide to Computer Network Security*, pp. 457–475, 2017.

[32] Y. Wen and H.-M. Wang, "A isolated execution model based on local virtualization technology," Ph.D. dissertation, 2008.

[33] R. P. Goldberg *et al.*, "Architectural principles for virtual computer systems," Ph.D. dissertation, Harvard University Cambridge, MA, 1973.

[34] E. L. C. Macedo *et al.*, "On the security aspects of Internetf of things: A systematic literature review," *J. Commun. Netw.*, vol. 21, no. 5, pp. 444–457, 2019.

[35] H. Cheng *et al.*, "Machine learning based low-rate DDoS attack detection for SDN enabled IoT networks," *International J. Sensor Netw.*, vol. 34, no. 1, pp. 56–69, 2020.

Haosu Cheng received a Ph.D. degree from Beihang University, Beijing, China in 2020. He is a Postdoctoral Researcher at the School of Software, Shandong University and Shandong CVIC Software Engineering Co., Ltd, Jinan, China. Concurrently, he serves as a Lecturer and Researcher at the Shandong Key Laboratory of Blockchain Finance, Shandong University of Finance and Economics, Shandong, China. His research interests include security of IoT, blockchain, and information security.

Guanquan Shi received the B.E. degree in Internet of Things Engineering from Changshu Institute of Technology, Suzhou, China in 2021. He is currently working toward the M.Sc. degree in Computer Science and Technology with Shandong University of Finance and Economics, Jinan, China. His research interests include blockchain, cross-chain and information security.

Kangkang Zhang , Ph.D., Associate Professor. He received a Ph.D. degree from Shandong University, Jinan, China in 2006. His research interests include software and data engineering, blockchain, digital media, intelligent information processing, etc.